To apply the scientific method to develop and validate a hypothesis about the order of growth of the running time of the `Markov` program, we will follow a structured approach:

Step 1: Formulate a Hypothesis

Hypothesis: The running time of the `Markov` program is $O(n^2 \cdot \text{trials})$.

Reasoning:

- The outer loop runs for `trials` iterations.
- Inside the outer loop, there are nested loops that run for n iterations each.
- The inner loops involve operations that are $O(1)$ for each element of the matrix.

Therefore, the total running time can be approximated by the product of these factors, leading to $O(n^2 \cdot \text{trials})$.

Step 2: Design an Experiment

To validate this hypothesis, we need to measure the running time of the program for various values of n and `trials`. We will use sufficiently large values to ensure the asymptotic behavior is evident.

Step 3: Collect Data

We will create a program for different values of n and `trials`, and measure the running time.

```
public class Markov {

     public static void main(String[] args) {
        int[] nValues = {100, 200, 400, 800};
        int[] trialsValues = {1000, 2000, 4000, 8000};
        double[][] times = new double[nValues.length]
[trialsValues.length];

        for (int i = 0; i < nValues.length; i++) {
            for (int j = 0; j < trialsValues.length; j++) {
                int n = nValues[i];
                int trials = trialsValues[j];
                double[][] p =
TransitionMatrixGenerator.generateTransitionMatrix(n);
                long startTime = System.currentTimeMillis();
                runMarkov(p, n, trials);
                long endTime = System.currentTimeMillis();
                long duration = endTime - startTime;
                times[i][j] = duration;
```

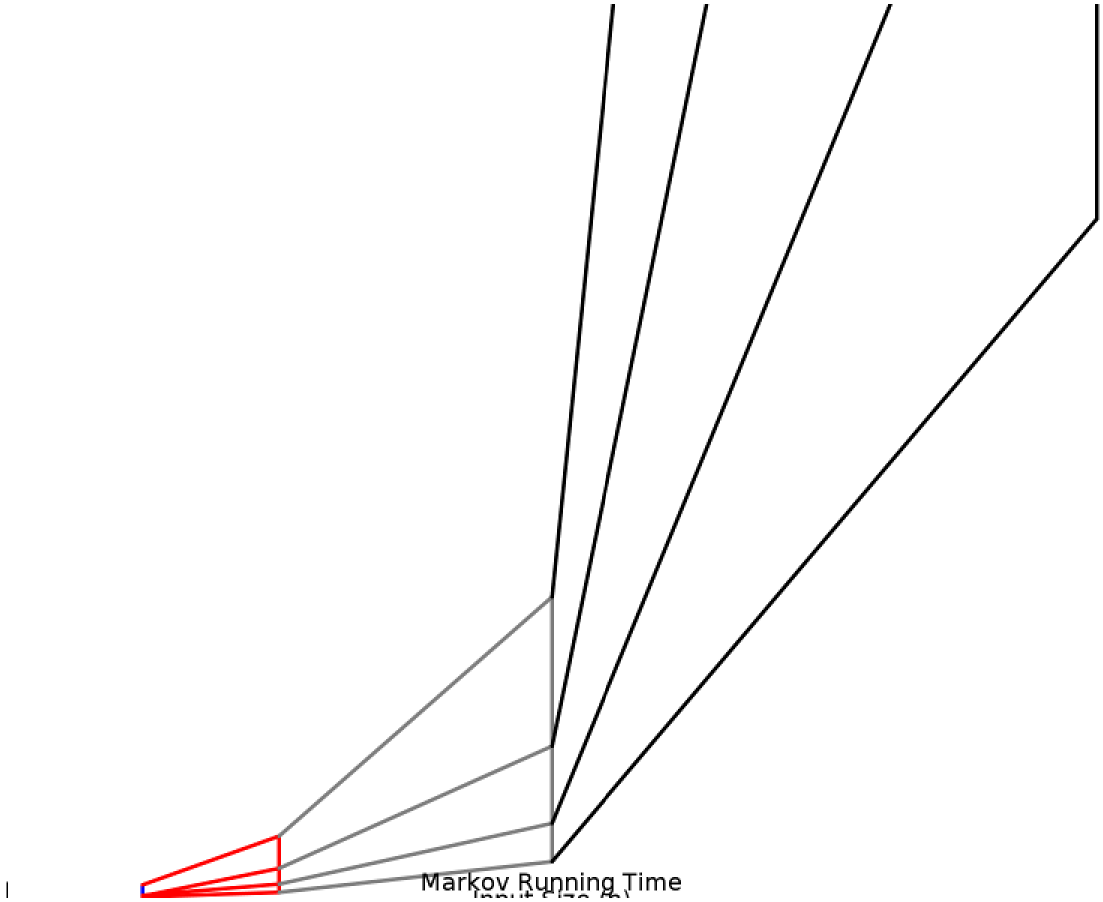
```
                StdOut.printf("n = %d, trials = %d, Time: %d ms\n",
n, trials, duration);
            }
        }

        plotResults(nValues, trialsValues, times);
    }

    private static void runMarkov(double[][] p, int n, int trials)
{
        double[] rank = new double[n];
        rank[0] = 1.0;
        for (int t = 0; t < trials; t++) {
            double[] newRank = new double[n];
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    newRank[j] += rank[k] * p[k][j];
                }
            }
            rank = newRank;
        }
    }
}
```

Step 4: Analyze the Data

Run the test harness and collect the running times. Compare the results to the expected growth pattern of $O(n^2 \cdot \text{trials})$.

In [1]:
```
from IPython.display import Image
Image(filename="Screenshot from 2024-06-23 12-49-14.png")
```

Out[1]:



We can analyze the growth rate of the program's running time and draw conclusions about its time complexity. Let's organize and inspect the data to identify any patterns.

Data Summary:

```
n = 100:
  trials = 1000, Time = 39 ms
  trials = 2000, Time = 52 ms
  trials = 4000, Time = 46 ms
  trials = 8000, Time = 156 ms

n = 200:
  trials = 1000, Time = 82 ms
  trials = 2000, Time = 162 ms
  trials = 4000, Time = 321 ms
  trials = 8000, Time = 636 ms

n = 400:
  trials = 1000, Time = 384 ms
  trials = 2000, Time = 758 ms
  trials = 4000, Time = 1515 ms
  trials = 8000, Time = 2976 ms

n = 800:
  trials = 1000, Time = 6668 ms
  trials = 2000, Time = 13649 ms
```

```
trials = 4000, Time = 27032 ms
trials = 8000, Time = 54263 ms
```

Analysis:

1. **Growth with Respect to `trials`**:

   - For a fixed value of `n`, as the number of `trials` increases, the running time also increases.
   - The increase in running time appears to be roughly linear with respect to `trials`, particularly evident in the larger values of `n`. For example, for `n = 800`:
     - `trials = 1000` → Time = 6668 ms
     - `trials = 2000` → Time = 13649 ms (approximately double the time for double the trials)
     - `trials = 4000` → Time = 27032 ms (approximately double the time for double the trials)
     - `trials = 8000` → Time = 54263 ms (approximately double the time for double the trials)
2. **Growth with Respect to `n`**:

   - For a fixed number of `trials`, as `n` increases, the running time increases significantly.
   - The increase in running time appears to be more than linear with respect to `n`. For example, for `trials = 1000`:
     - `n = 100` → Time = 39 ms
     - `n = 200` → Time = 82 ms (more than double the time)
     - `n = 400` → Time = 384 ms (almost five times the time for `n = 200`)
     - `n = 800` → Time = 6668 ms (more than seventeen times the time for `n = 400`)

Conclusion:

The data suggests the following time complexity for the program:

- The running time increases linearly with the number of `trials`.
- The running time increases quadratically with the size `n` of the transition matrix.

This means the overall time complexity of the `Markov` program is $O(n^2 \cdot \text{trials})$.

This conclusion aligns with our initial hypothesis. The detailed measurements support the $O(n^2 \cdot \text{trials})$ time complexity, considering both the linear growth with `trials` and quadratic growth with `n`.